# Field-based Motion Coordination in Pervasive Computing Scenarios

Marco Mamei and Franco Zambonelli

Dipartimento di Scienze e Metodi dell'Ingegneria,
University of Modena and Reggio Emilia
Via Allegri 13, 42100 Reggio Emilia, Italy
{mamei.marco, franco.zambonelli}@unimo.it

## 1 Introduction

As computing is becoming pervasive, autonomous computer-based systems are going to be embedded in all everyday objects and in the physical environment. In such a scenario, mobility too, in different forms, will be pervasive. Mobile users, mobile devices, computer-enabled vehicles, as well as mobile software components, define an open and dynamic networked world that will be the stage of near-future software applications.

Traditionally, software applications have always been built by adopting programming paradigms rooted on not-autonomous components coupled at design time by fixed interaction patterns. Although simple in principle, this approach seems inadequate to deal with the dynamism of the above sketched application scenario and it is likely to lead to even-more brittle and fragile applications.

Only in recent years, the research on software agents has fostered new programming paradigms based on autonomous components (i.e. components with a separated thread of execution and control) interacting to realize an application [3, 6, 25]. In this chapter, the term agent can refer not only to software components, but also to any autonomous real-world entity with computing and networking capability (e.g., a user carrying on a Wi-Fi PDA, a robot, or a modern car). Autonomous interacting components are intrinsically robust, in that for example, if a component breaks down or goes out of the wireless network, the others can autonomously and dynamically re-organize their interaction patterns to account for such a problem. The key element leading to such robust, scalable and flexible behaviors is coordination: the agents' fundamental capability to dynamically decide on their own actions in the context of their dynamic operational environment. Autonomous components, in fact, must be able to coordinate their activities' patterns to achieve goals, that possibly exceed their capabilities as singles, despite - and possibly taking advantage - of environment dynamics and unexpected situations [24, 31].

Unfortunately, general and widely-applicable approaches to program and manage these autonomous interacting systems are unknown. The main conceptual difficulty is that it is possible to apply a direct engineered control only on the single agents' activities, while the application task is often expressed at the

global-ensemble scale [4, 24]. Bridging the gap between single and global activities is not nearly easy, but it is although possible: distributed algorithms for autonomous sensor networks have been proposed and successfully verified [9], routing protocols in MANET (in which devices coordinate to let packets flow from sources to destinations) have been already widely used [26]. The problem is still that the above successful approaches are ad-hoc to a specific application domain and it is very difficult to generalize them to other scenarios. There is a great need for general, widely applicable engineering methodologies, middleware and APIs to embed, support and control coordination in multi-agent systems [1, 27, 17, 32].

A significant application scenario related to the above general problem is distributed motion coordination. Here the task is of coordinating the movements of a large set of autonomous agents in a distributed environment. The term agent can refer not only to software components, but also to any autonomous real-world entity with computing and networking capability (e.g., a user carrying on a Wi-Fi PDA, a robot, or a modern car). The goals of agents' coordinated movements can be various: letting them meet somewhere, distribute themselves accordingly to specific spatial patterns, or simply move in the environment without interfering with each other and avoiding the emergence of traffic jams.

With this regard, protocols based on the concept of force fields (field-based coordination) [15, 16, 18] suit well the needs of distributed motion coordination. These approaches are inspired by the physical world, i.e., from the way masses and particles in the universe move and globally self-organize driven by gravitational and electro-magnetic fields.

In these protocols, the agents' operation environment is abstracted and described by means of fields capable of expressing forces driving agents' activities. Agents can locally perceive these fields and decide where to go on the basis of which fields they sense and their magnitudes.

Specifically, on the one hand, it is possible to assume that each agent can generate application-specific fields, conveying some information about the local environment and/or about itself. Fields will be propagated in the environment according to field-specific laws and to be possibly composed with other fields. On the other hand, agents can perceive these fields and move accordingly (or, if the agent is a program running on a mobile device, it can suggest its user how to move), e.g., by following a field gradient downhill or uphill. Since agents' movements are simply driven by these abstract force fields, without any central controller, the global coordination emerges in a self-organized way from the interrelated effects of agents following the fields' shape and of dynamic fields re-shaping due to agents' movements.

Engineering a coordination protocol within a field-based approach is basically a bottom-up approach and consists in specifying how agents generate fields, how these fields are propagated, and how agents subscribe and react to the fields. The result is a robust specification that automatically adapts to changing environmental conditions that require limited computational efforts by agents, and only strictly local communication efforts. The analogy of field-based protocols

and physical laws carries on a further advantage: field-based coordination activities can be effectively modeled in dynamical systems terms, i.e., via differential equations that can be easily numerically integrated for the sake of rapid verification.

The continue of this chapter is structured as follows: Section 2 describes the problem of motion coordination along with its possible applications and its main requirements. Section 3 presents field-based solutions, surveying systems that exploited this approach. Section 4 actually shows how field-based coordination can be modeled, in terms of a dynamical system formalism. Section 5 deals with programming field based approaches, presenting a reference middleware and programming model. Finally, Section 6 outlines open issues and research directions.

## 2 Adaptive Motion Coordination: Applications, Issues and Requirements

Enabling a set of agents to coordinate their movements in an environment and engineering a coordination protocol accordingly is not a problem if: *(i)* agents know each other; *(ii)* they have a global knowledge about the environment in which they situate; *(iii)* they are not resource constrained. In these cases, it is possible to define and optimize any needed protocol to orchestrate agents' movements, and it is possible for agents to communicate with each other and to exploit environment knowledge as needed to implement such protocol. Unfortunately, the above assumptions cannot be made in the design and development of the software support for a mobile and wireless computing scenario, where: *(i)* the number and the identities of agents, together with their operational environment are likely to vary in time, and the software systems must be engineered so as adaptively deal with these situations; *(ii)* the need for scalability forbids a global perception of the environment and instead calls for a strictly-local view, enabling the system to scale linearly with its size; *(iii)* agents are likely to be executed on resource-limited devices. This implies the need for cost-effective solutions that should not require the execution of complex algorithmic task or of complex communication protocols.

The challenge of identifying adaptive and effective protocols for the engineering of motion coordination activities in mobile computing scenarios is strictly tied to the problems of having agents be "context-aware". Coordination, by its own nature, requires some sort of context awareness: an agent can coordinate with other agents only if it is aware of "what is around". In mobile computing scenarios, however, such contextual information must be dynamically obtained by agents, so as to enable adaptivity; simple and cheap to be obtained and expressive enough to facilitate the definition of coordination activities and their execution by agents.

It is worth noting that the focus in not on how contextual information can be produced (e.g., it is possible to assume that localization mechanisms are available [11]), but how and in which form it can be made available to agents.

The following of this section introduces a simple case study and show that current coordination models are, form a software engineering point of view, inadequate to effectively address motion coordination problems.

### 2.1 A Case Study Scenario

The presented case study involves the problem of providing support for tourists visiting a museum. Specifically, it focuses on how tourists can be supported in planning their movements across a possibly large and unfamiliar museum and in coordinating such movements with other, possible unknown, tourists. It is possible to assume that each tourist is provided with a wireless-enabled computer assistant (e.g., a PDA) providing support for motion coordination (i.e. suggests the user on where to go). The kind of coordination activities involved may include:

- scheduling attendance at specific exhibitions occurring at specific times. Here the application goal is simply to route tourists across the museum to let them reach the right place on time.
- having a group of students split in the museum according to teacher-specific laws. Also in this case, the application goal is guide the students to let them visit relevant rooms accordingly to a specified schedule.
- letting a group of agents (e.g. museum security guards) to move maintaining a formation. Here the PDAs could instruct guards to maintain a specified distance form each other so as to improve museum coverage.
- helping a tourist to avoid crowd or queues while visiting the museum. Here the PDA can route the tourist to avoid a crowded corridor, or can change the visit schedule so as to postpone rooms that are overcrowded at the moment.
- support a group of user to meet in the museum: meet in a specific room, join a specific person, meet in the most convenient room, accordingly to the participants actual location. Here the PDA would lead the user to the meeting location.
- in case of emergency, the system can enforce a evacuation plan, guiding tourists to nearby emergency exits, while avoiding crowds.

Accordingly with the assumptions made before, it is possible to assume that: *(i)* the museum environment is unknown and dynamic: the museum floor-plan may be unknown, agents can enter and leave the museum at will and they are free to roam across the building. Agents are enabled to interact one another either directly or via an embedded infrastructure, possibly installed in the building. Since agents are not provided with any a-priori information on the museum floor-plan or crowd condition, they have to acquire relevant information dynamically in-place, by interacting with other agents or with available services. *(ii)* they are able to perceive only a local view on the museum. This is both to support the scalability of the system and *(iii)* not to overload the resource-limited PDAs with large amount of information.

It is worth noting that the above scenario and the associated motion coordination problems are of a very general nature, being isomorphic to scenarios

such as, e.g., traffic management and forklifts activity in a warehouse, where navigators' equipped vehicles hint their pilots on where to go, or software agents exploring the Web, where mobile software agents can coordinate distributed researches by moving on various Web-sites. Therefore, also all the considerations presented in this chapter are of a more general validity, besides the addressed case study.

### 2.2 Inadequacy of Traditional Approaches

Most coordination models and middleware used so far in the development of distributed applications appear inadequate in supporting coordination activities in mobile and wireless computing scenarios.

In the following paragraphs will be presented various coordination models and middleware to illustrate their inadequacies form a software engineering perspective. The analysis will be mainly focused on evaluating how those middleware provide agents with contextual information and whether if the information provided is suitable for the motion coordination task.

Models based on *direct communication* promote designing a distributed application by means of a group of components that are in charge to communicate with each other in a direct and explicit way. Systems like Jini [13] and FIPA-based agent systems [3] are examples of middleware infrastructures rooted on a direct communication model. The problem of this approach is that agents are placed in a "void" space: the model, per se, does not provide any contextual information. Agents can only perceive and interact with other components/agents, and the middleware support is mostly reduced to helping in finding communication partners. Thus, each agent has to "manually" become context aware by discovering and explicitly interacting with the other entities in the environment. For instance, in the case study, an agent would have to explicitly retrieve and communicate with some services to gain the museum map, it would have to discover which other tourists are currently populating the museum, and explicitly negotiate with them to agree on a specific motion coordination policy (e.g., to move avoiding the formation of queues or to meet each other at a suitable place). From a software engineering perspective, this imposes a notable burden in the agent code and typically ends up with ad-hoc, not scalable, and not adaptable solutions.

Models based on *shared data-spaces* support inter-agent interactions through shared localized data structures. These data structures can be hosted in some data-space (e.g., tuple space), as in EventHeap [14], JavaSpaces [10], or they can be carried on by agents themselves and dynamically merged to enable interactions, as in Lime [25] or XMiddle [21]. In these cases, agents are no longer placed in a totally void space, but live in an environment that can be modeled and described in terms of the information stored in the data spaces. Such information, typically referring to local conditions, can provide some sort of context-awareness to agents without forcing them to directly communicate with each other. For instance, in the case study, one can assume to have, at each room and corridor, a local data-space storing both local map information and messages left by the

other agents about their presence and possibly about their intended next position. Still, to enforce a specific motion coordination pattern, agents may have to access several data-spaces to access all the required information (e.g., which agents are in which rooms), to build an internal representation of the current situation (e.g., where crowds and queues are) and then decide the next movements by (again) negotiating with other agents or accordingly to a predefined, non adaptable, set of rules. The result is again that lots of code and computational effort is required to "understand" and exploit the contextual information for the achievement of a specific motion coordination task. This makes also this approach ineffective from a software engineering perspective.

*Event-based models* relying on *publish/subscribe* mechanisms make agents interact with each other by generating events and by reacting to events of interest, without having them to interact explicitly with each other. Typical infrastructures rooted on this model are: Jedi [8] and Siena [7]. Without doubt, event-based model promotes stronger context-awareness, in that components can be considered as embedded in an active environment able of notifying them about what's happening around, freeing agents from the need of explicitly querying other agents or the environment (as in direct-communication and data-space models) to discover what's happening. For instance, in the case study, a possible use of this approach would be to let each agent notify its movements across the museum, update (accordingly to other agents' notified movements) its internal representation of the crowd distribution in the museum (i.e. context-awareness) and then move properly by continuously adapting its plans accordingly to the newly received events. However, from a software engineering perspective, the information conveyed by events tends to be too low-level to be effectively used by agents, forcing them to catch and analyze a possibly large number of inter-related events to achieve a specific motion coordination pattern. This process is of course a big burden in the agent code, making also such kind of middleware unsuitable from a software engineering point of view.

## 3 Field-based Approach to Motion Coordination

The common general problem, of the above interaction models, is that they are typically used to gather/communicate only a general purpose, not expressive, description of the context. This general purpose representation tends to be strongly separated from its usage by the agents, typically forcing them to execute complex algorithms to elaborate, interpret and decide what to do with that information. In the above mentioned meeting application, for example, agents could be provided with the museum floor-plan and the position of other agents in the meting group. Still, even if that information is a *complete* representation of their operational environment, agents will have to execute complex algorithms to decide and negotiate where to meet and how to go there.

On the contrary, if the context would have been represented expressively to facilitate agents in the achievement of a specific task, agents would trivially use that information to decide what to do. For example, in the above meeting appli-

cation, if the agents would be able to perceive in their environment something like a 'red carpet' leading to the meeting room, it would be trivial for them to exploit the information: just walk on the red carpet!

So the point is: how is it possible to create the 'red carpet'? How to effectively represent context for the sake of motion coordination?

An intriguing possibility is to take inspiration from the physical world, and in particular from the way masses and physical particles move and globally self-organize their movements accordingly to that local contextual information that is represented by gravitational and electro-magnetic fields. These fields are sorts of 'red carpets': particles achieve their tasks simply by following the fields.

This is at the basis of the field-based approach. Following this approach, agents achieve their goals not because of their capabilities as single individuals, but because they are part of an (auto)organized system that leads them to the goals achievement. Such characteristics also imply that agents' activities are automatically adapted to the environmental dynamic, which is reflected in a changing view of the environment, without forcing agents to re-adapt themselves.

### 3.1 Key Concepts in the Field-based Approach

Field-based approach is mainly driven by the above analysis and aims at providing agents with abstract - simple yet effective - representations of the context. To this end, the field-based approach delegates to a middleware infrastructure or to the agents themselves (connected in a peer-to-peer mobile ad-hoc network) the task of constructing and automatically updating an essential distributed "view" of the system situation - possibly tailored to application-specific motion coordination problems - that "tells" agents what to do (i.e., how to move to implement a specific motion coordination patterns). Agents are simply let with the decision of whether to follow such a suggestion or not. Operatively, the key points of the field-approach can be schematized in the following four points:

1. The environment is represented and abstracted by "computational fields", spread by agents or by the infrastructure. These fields convey useful information for the agents' coordination tasks and provide agents with strong coordination-task-tailored context awareness;
2. The coordination protocol is realized by letting agents move following the "waveform" of these fields.
3. Environmental dynamics and agents' movements induce changes in the fields' surface, composing a feedback cycle that influences agents' movement (point 2).
4. This feedback cycle let the system (agents, environment and infrastructure) auto-organize, so that the coordination task is eventually achieved.

A computational field can be considered a sort of distributed data structure characterized by a unique identifier, a location-dependent numeric value. Fields are locally accessible by agents depending on their location, providing them and a local perspective of the global situation of the system. For instance, with reference to the case study, the guide of the museum can spread in the environment

a computational field whose value monotonically increase as it gets farther from his/her (See figure 1), and thus implicitly enabling an agent, from wherever, of "sensing" how distant the guide is and where (in which direction) (s)he is.

Fields can be generated by the agents or by the environment, and their existence must be supported by a proper infrastructure. From the agent point of view, it does not matter if the infrastructure is implemented in some external servers accessed by the agent, or is part of the agent itself. What that matters is that it provide an up-to-date, local, field-based representation of the environment.

In field-based protocols, the simple principle to enforce motion coordination is by having agents move following the local shape of specific fields. For instance, a tourist looking for a guide can simply follow downhill the corresponding computational field.

Dynamic changes in the environment and agents' movements induce changes in the fields' surface, producing a feedback cycle that consequently influences agents' movement. For instance, should the museum guide be moving around in the museum, the field-supporting infrastructure would automatically update the corresponding computational field and would, consequently, have any agent looking for a guide re-adapt its movement accordingly. Should there be multiple guides in the museum, they could decide to sense each other's fields so as to stay as far as possible from each other to improve their reachability by tourists, and possibly dynamically re-shaping their formation when a guide, for contingent problems, has to move or go away.

Further examples follow later on. Here, it is worth noting that a field-based based system - following its physical inspiration - can be considered as a simple dynamical system: agents are seen as balls rolling upon a surface, and complex movements are achieved not because of the agents will, but because of dynamic re-shaping of this surface.

In a given environment, several different types of fields can exist and be propagated by agents and by the infrastructure, accordingly to field-specific laws. Of course, the agents and the infrastructure can decide to propagate any type of application-specific fields, accordingly to propagation rules that can facilitate the achievement of application-specific problems.

The achievement of an application-specific coordination task, however, is rarely relying on the evaluation, as it is, of an existing computational field (as in the case of a tourist looking for a guide and simply following the specific field of that guide). Rather, in most cases, an application-specific task relies on the evaluation of an application-specific *coordination field*, as a combination (e.g., linear) of some of the locally perceived fields. The *coordination field* is a new field in itself, and it is built with the goal of encoding in its shape the agent's coordination task. Once a proper *coordination field* is computed, agents can achieve their coordination task by simply following (deterministically or with some probabilistic rule) the shape of their *coordination field* uphill, downhill, or along its equipotential lines (depending on the specific problem) as if they were walking upon the *coordination field* associated surface (see figure 1). For

instance, in the case study, for guides to stay as far as possible from each other, they have to follow uphill a *coordination field* resulting from the combination of all the computational fields of each guide.

In the following, the *coordination field* will be referred as the only field (eventually computed by the agent) driving the agent movements.

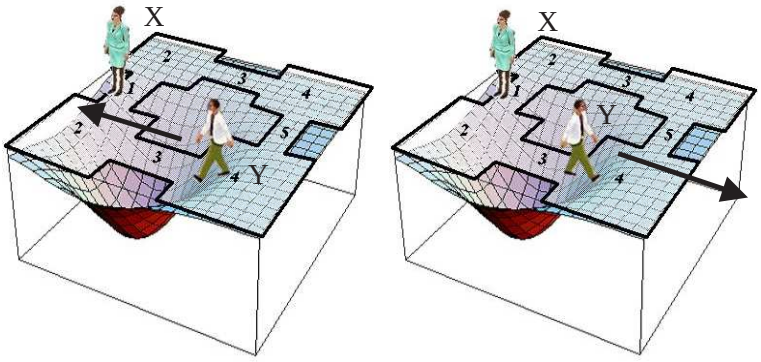


**Fig. 1.** (left) The tourist Y senses the field generated by the tourist guide X (increasing with the distance from the source, as the numeric values in each room indicates). (left) Y agent follows the field downhill to approach X; (right) Y follows the field uphill to get farther from X. In this figure, agent Y uses the field generated by X directly as an application-specific *coordination field*.

### 3.2 The State of the Art in Field-based Coordination

Several proposals addressing the problem of distributed motion coordination with field-based protocols have been proposed in the last few years. The basic idea to implement the concept of a field is to rely on distributed data structure actually spread in the environment. Such distributed data structures must be supported by a distributed infrastructure. Specifically, a field would be injected in the network from a specific node. Then it will be propagated across the infrastructure hop-by-hop. During propagation the field's value would be changed so as to let the field assume the intended shape.

In the case study, it is possible to assume the presence, in the museum, of a densely distributed network of computer-based devices, associated with rooms, corridors, art pieces, alarm systems, climate conditioning systems, etc. All these devices will be capable of communicating with each other and with the mobile devices located in their proximity via the use of a short-range wireless link. Moreover, it is important to make the following two assumptions: *(i)* devices are provided with localization mechanisms enabling them to know neighbors' coordinates in a private local coordinate frame [11]. *(ii)* The rough topology

of the ad-hoc network being formed by computer-based devices resemble the museum's topology (i.e. floor-plan). This means in particular that there are not network links between physical barriers (like walls). Several mechanism may be employed to achieve this property, it can be assumed that either the devices are able to detect and drop those network links crossing physical barriers (e.g. relying on signal strength attenuation or some other sensor installed on the device) or that the museum building is pre-installed with a network backbone - reflecting its floor-plan topology - and that all the nodes can connect only to the backbone. These two properties are needed to assure that the fields are propagated coherently with the museum floor-plan (i.e. agents can follow fields without stumbling into walls!).

Co-Fields [18] is the model developed by the authors of this chapter and is the one that closely targets the application scenario described so far. This model has been implemented upon the Tuples On The Air (TOTA) [20] middleware infrastructure. The TOTA middleware would be installed on a large number of the above devices, that will connect in an ad-hoc network suitable to support distributed data structures implementing the field abstraction. Agents would connect to this infrastructure, inject field-like data structures that propagate hop-by-hop across the infrastructure, sense locally the stored data structures and move accordingly.

Similar to Co-Fields is the Multi-layered Multi Agent Situated System (MMASS) formal model for multi-agent coordination, described in [2]. This model represents the environment as a multi-layered graph in which agents can spread abstract fields representing different kinds of stimuli through the nodes of this graph. The agents' behavior is then influenced by the stimuli they perceive in their location. In fact agents can associate reactions to these stimuli, like in an event-based model, with the add-on of the location-dependency that is associated to events and reactions. The main application scenarios of this model is quite different from the one discussed in this chapter since it focuses on simulation of artificial societies and social phenomena. For this reason MMASS lacks of an implemented infrastructure supporting the model and it is only realized via simulation based on cellular automata.

An area in which the problem of achieving effective context-awareness and adaptive coordination has been addressed via a field-based approach is Amorphous Computing [**?**,23]. The particles constituting an amorphous computer have the basic capabilities of propagating sorts of abstract computational fields in the network, and to sense and react to such fields. In particular, particles can transfer an activity state towards directions described by fields' gradients, so as to make coordinated patterns of activities emerge in the system independently of the specific structure of the network. Such mechanism can be used, among other possibilities, to drive particles' movements and let the amorphous computer self-assemble in a specific shape.

A very similar approach to self-assembly has been proposed in the area of modular robots [28]. A Modular robot is a collection of simple autonomous actuators with few degrees of freedom connected with each other. A distributed

control algorithm is executed by all the actuators that coordinate to let the robot assume a global coherent shape or a global coherent motion pattern (i.e. gait). Currently proposed approaches [28] adopts the biologically inspired idea of hormones to control such a robot. Hormone signals are similar to fields in that: they propagate through the network without specific destinations, their content can be modified during propagation and they may trigger different actions for different receivers.

Shifting from physical to virtual movements, the popular videogame "The Sims" [29] exploits sorts of computational fields, called "happiness landscapes" and spread in the virtual city in which characters live, to drive the movements of non-player characters. In particular, non-player characters autonomously move in the virtual Sims city with the goal of increasing their happiness by climbing the gradients of specific computational fields. For instance, if a character is hungry, it perceives and follows a happiness landscape whose peaks correspond to places where food can be found, i.e., a fridge. After having eaten, a new landscape will be followed by the character depending on its needs.

It is interesting to report that there are also systems that adopt a field-based approach, without requiring the presence of any any needed distributed infrastructure. The field-based representation, in fact, is computed internally by the agents by exploiting data coming from sensors (typically a camera) installed on the agents themselves. For example, an agent seeing an obstacle through its camera would represent the obstacle as a repealing field. The main advantage of this approach is that it does not require any communication between the agents. The drawback is the need of complex sensors to be mounted on agents (e.g. the camera) and the need for complex algorithms mapping images into fields. Examples of this kind of approach can be found in robotics.

The idea of fields driving robots movement is not new [16]. For instance, one of the most recent re-issue of this idea, the Electric Field Approach (EFA) [15], has been exploited in the control of a team of Sony Aibo legged robots in the RoboCup domain. Following this approach, each Aibo robot builds a field-based representation of the environment from the images captured by its head mounted cameras (stereo-vision), and decides its movements by examining the fields' gradients of this representation.

Apart from the presented differences, the modeling of the field-based approach in both the above two kinds of models is rather similar and it basically consists in representing the environment by means of the fields and having each agent to follow the gradient of a *coordination field* computed by combining fields relevant for its application task.

## 4 Modeling Field-based Coordination

The physical inspiration of the field-based approach invites modeling field-based protocols in terms of a dynamical system. Specifically, this implies modeling fields as continuous functions and writing the differential equations governing the motion of a point (the agent) driven by the gradient of a specific *coordi-*

*nation field* (the field driving agents movements, evaluated by the agent, by locally combining other fields in the environment). Once the analytical shape of the *coordination field* is defined, writing and solving numerically the differential equations of the system is rather simple. This provides an effective way to simulate the system, and it can be regarded as a tool to make experiments, to quickly verify that a *coordination field* correctly expresses a coordination task, and to tune coefficients.

More in detail, it is rather easy to see that considering the agent $i$, denoting its coordinates (for sake of notation simplicity restricted to the 2-D case) as $(x_i(t), y_i(t))$ and its *coordination field* as $CF_i(x, y, t)$, the differential equations governing $i$ behavior are in the form:

$$\begin{cases} \frac{dx_i}{dt} = v\frac{\partial CF_i(x,y,t)}{\partial x}(x,y,t) \\ \frac{dy_i}{dt} = v\frac{\partial CF_i(x,y,t)}{\partial y}(x,y,t) \end{cases}$$

If $i$ follows uphill the *coordination field*. They are:

$$\begin{cases} \frac{dx_i}{dt} = -v\frac{\partial CF_i(x,y,t)}{\partial x}(x,y,t) \\ \frac{dy_i}{dt} = -v\frac{\partial CF_i(x,y,t)}{\partial y}(x,y,t) \end{cases}$$

If $i$ follows downhill the *coordination field*.

This is because the direction of the gradient of the agent's *coordination field*, evaluated towards the spatial coordinates, points to the direction in which the *coordination field* increases. So the agent $i$ will follow this gradient or will go in the opposite direction depending on if it wants to follow its *coordination field* uphill or downhill. $v$ is a term that can model the speed of an agent.

The case in which the agent follows an equipotential line of the coordination space, is slightly more complicated, because if the space dimension is greater than 2, it is not possible to talk about an equipotential line, but only about an equipotential hyper-surface. This hyper surface will be the one orthogonal to the gradient of the *coordination field* evaluated only towards the space coordinates, which will be thus a function of the time:

$$\nabla CF_i(x_1, x_2, ..., x_n)(t)$$

So the only dynamical equation it is possible to write in this case, is the one that specifies that the agent's movements belong to this hyper surface:

$$(\frac{dx_1^i}{dt}, \frac{dx_2^i}{dt}, ..., \frac{dx_n^i}{dt}) \cdot \nabla CF_i(x_1^i, x_2^i, ..., x_n^i)(t) = 0$$

Where the *dot* stands for the classical scalar product.

In the following of this section field-based protocols and the dynamical system formalism will be applied to a number of motion coordination problems. Before proceeding, it is worth noting that, considering those situations in which agent movements do not occur in an open space but are constrained by some environmental conditions, the dynamical system description become more complex. The definition of "constrained" equations would involve either some artificial force

fields to constrain agents to stay within the building plan or a domain not based on $\Re^n$, but on a more general and complex manifold. Due to the complexities involved in the description of such equations, this chapter considers and discusses field-based protocols only in open, unconstrained spaces, an approach which is anyway valuable to evaluate the correctness of a coordination protocol.

However, to overcome this limitation, a multi-agent simulation of the museum application has been used to complement the dynamical system description. Such simulation has been developed using the Swarm toolkit [30] and enables to model: any required spatial environment (e.g., a specific museum map); the presence in such environment of any needed number of fields; and the presence of any needed number of agents each its own goals (e.g., number of tourists each with a specific plan of visit to the museum).

### 4.1 Flock Field: Moving Maintaining a Formation

Consider the case study of having agents distribute according to specific geometrical patterns ("flocks"), and to let them preserve such patterns while moving. More specifically, agents can represent security guards (or security robots) in charge of cooperatively monitor a building by spreading in the building so as to stay at specified distances from each other [12]. To this end, it is possible to take inspiration from the work done in the swarm intelligence research [4]: flocks of birds stay together, coordinate turns, and avoid each other, by following a very simple swarm algorithm. Their coordinated behavior can be explained by assuming that each bird tries to maintain a specified separation from the nearest birds and to match nearby birds' velocity. To implement such a coordinated behavior with a field-based protocol and apply it to the case study, each agent can generate a field *(flock-field)* whose magnitude assumes the minimal value at specific distance from the source, distance expressing the intended spatial separation between agents. The final shape of this field approaches the function depicted in figure 2-top. Fields are always updated to reflect peers' movements. To coordinate movements, peers have simply to locally perceive the generated fields, and to follow downhill the gradient of the fields. The result is a globally coordinated movement in which peers maintain an almost regular grid formation (see figure 2-bottom).

Analytically, the *(flock-field)* generated by an agent $i$ located at $(X_P^i, Y_P^i)$ can be simply described as follows:

$$d = \sqrt{(x - X_P^i)^2 + (y - Y_P^i)^2}$$

$$FLOCK_i(x, y, t) = d^4 - 2a^2d^2$$

where $a$ is again the distance at which agents must stay away from each other. Starting from these simple equations, one can write, for each of the agents in the set, the differential equations ruling the dynamic behavior of the system, i.e., expressing that agents follow downhill the minimum of the *(flock-field)*. These have the form:

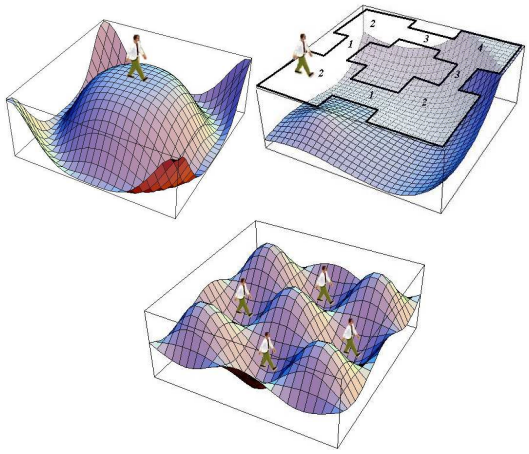

**Fig. 2.** (top) Ideal shape of the flock field. (bottom) when all the agents follow other agents' fields they collapse in a regular grid formation.

$$\begin{cases} \frac{dx_i}{dt} = -v\frac{\partial CF_i(x,y,t)}{\partial x}(x_i, y_i) & i = 1, 2..., n \\ \frac{dy_i}{dt} = -v\frac{\partial CF_i(x,y,t)}{\partial y}(x_i, y_i) & i = 1, 2..., n \end{cases}$$

$$\begin{cases} \frac{dx_i}{dt} = -v\frac{\partial min(FLOCK_1, FLOCK_2, ..., FLOCK_n)}{\partial x}(x_i, y_i) & i = 1, 2..., n \\ \frac{dy_i}{dt} = -v\frac{\partial min(FLOCK_1, FLOCK_2, ..., FLOCK_n)}{\partial y}(x_i, y_i) & i = 1, 2..., n \end{cases}$$

Such equations can be numerically integrated by making use of any suitable mathematical software. In this study, the Mathematica package [22] has been used. Figure 3(a) shows the results obtained by integrating the above equations, for different initial conditions, and for a system composed by four agents. The figure shows a $x$-$y$ plane with the trajectories of the agents of the system (i.e. the solutions of $(x_i(t), y_i(t))$ evaluated for a certain time interval) while moving in a open space. It is rather easy to see that the four agents maintain a formation, trying to maintain specified distances form each other. Figure 3(b) shows the result of the motion coordination in the simulated environment. Specifically, a simple museum map has been draw, with three agents (the white dots) moving in it. The formation consists in having the agents remain in adjacent rooms from each other.

### 4.2 Person Presence Field: Surrounding a Prey

The aim of this coordination task is to allow a group of agents ("predators") moving in the building to surround and eventually capture another agent ("prey"). As an example, one may, think at a group of security guards in charge of searching and catching a child that got lost in a big building and that is frightened

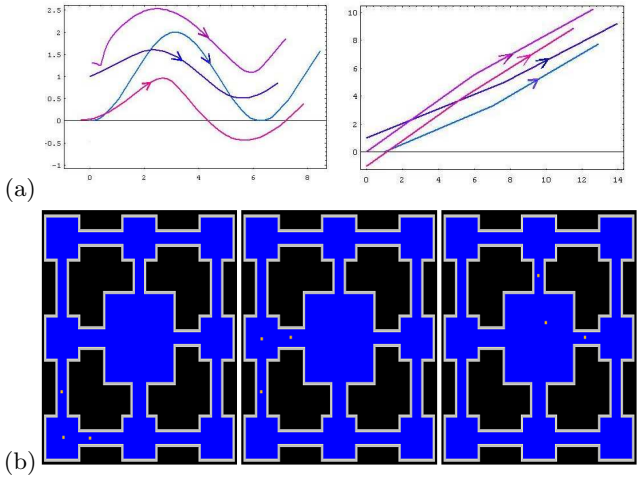


**Fig. 3.** (a) Solutions of the flock fields' differential equations, for different initial conditions. (b) Flocking: from left to right, different stages in the movement of a group of agents through the building and coordinating with each other so as to stay always in neighbor rooms.

by those big military men. This task of surrounding a prey relies on a field that must be generated by every agent in the building. This field will be referred as the *person presence* field to stress the fact it represent the presence of a person in the building. The *person presence* field simply has value that increases monotonically as the distance from the person increases. The analytical description of this field is straightforward. If the agent $i$ is at the coordinates $(X_P^i, Y_P^i)$, then it generates a *person presence* field whose equation is (see figure 4):

$$PRES_i(x, y, t) = 1 + k_p - k_p e^{-h_p((x-X_P^i)^2+(y-Y_P^i)^2)}$$

$$k_p, h_p > 0; \qquad 1 + k_p - k_p e^{-2h_p} \approx 1$$

Now, considering the prey's *coordination field* consisting in the linear combination of all the predators' fields, by following the decrease of the resulting field, the prey runs away from the predators.

$$CF^{prey}(x, y, t) = \sum_{i=1}^{n} -PRES_i^{pred}(x, y, t)$$

$PRES_i^{pred}$ is the *person presence* field of the predator agent $i$. Similarly, considering each predator's *coordination field* consisting of the linear combination between the prey's field and all the other predators' fields a predator is directed towards the prey, but avoids other predators.

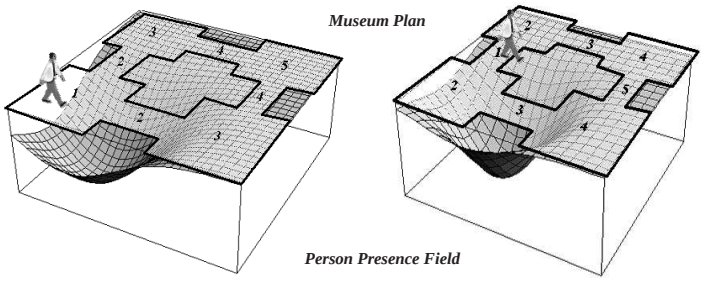


**Fig. 4.** Person Presence Fields in a building.

$$CF_i^{pred}(x,y,t) = PRES^{prey}(x,y,t) + \sum_{j=1,j\neq i}^{n} -PRES_i^{pred}(x,y,t)$$

$PRES^{prey}$ is the *person presence* field of the prey agent. Obtaining the differential equations governing the system is now just a matter of substituting the fields' analytical description together with the *coordination field*'s description in the differential equations describing the field-based protocol. The substitution will be reported just for this first example.

$$\begin{cases} \frac{dx^{prey}}{dt} = -v^{prey}\frac{\partial CF^{prey}(x,y,t)}{\partial x}(x^{prey}, y^{prey}) \\ \frac{dy^{prey}}{dt} = -v^{prey}\frac{\partial CF^{prey}(x,y,t)}{\partial y}(x^{prey}, y^{prey}) \\ \frac{dx_i^{pred}}{dt} = -v^{pred}\frac{\partial CF_i^{pred}(x,y,t)}{\partial x}(x_i^{pred}, y_i^{pred}) \qquad i = 1,2...,n \\ \frac{dy_i^{pred}}{dt} = -v^{pred}\frac{\partial CF_i^{pred}(x,y,t)}{\partial y}(x_i^{pred}, y_i^{pred}) \qquad i = 1,2...,n \end{cases}$$

The results of the numerical integration of these equations in the case of three predators and one prey are displayed in the following figure 5(a), which shows a common x-y plane with the trajectories of the elements of the system (i.e. the solutions of $(x_i(t), y_i(t))$ evaluated for a certain time interval). Here it possible to notice (figure 5(a)-left) that if the predators do not repeal one another they are not able to surround the prey and all reach the prey from the same direction. On the contrary (figure 5(a)-right), if they repeal each other, they reach the prey from different directions, surrounding it. Figure 5(b) shows the result of the motion coordination in the simulated environment. Maintaining a certain distance from each other, predator agents surround the prey without leaving any escape path.

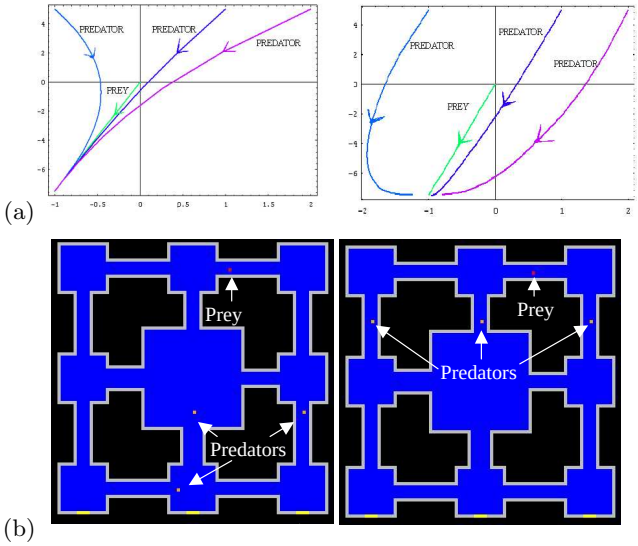


**Fig. 5.** (a) Surrounding a Prey: predators that simply follow the prey without enacting any coordination strategy are not able to surround the prey (top); predators being repulsed by other predators' fields are able to surround the prey (bottom). (b) surrounding in the simulator

### 4.3 Room Field and Crowd Field: Load Balancing

The aim of this coordination task is to allow the users to avoid queues while visiting the museum. For this reason their agents will drive them to visit the rooms in their schedule trying to avoid crowded areas. At the global scale, this realize a load balancing policy between users and museum's rooms. To this end other two fields need to be introduced: the *room field* and the *crowd field*. The *room field* is generated by every building's room, it has a value that increases with the distance from the source. The analytical description of this field is straightforward: considering the room $i$ to be located - the center of the room is taken as a reference - at the coordinates $(X_R^i, Y_R^i)$, then it is possible to describe its *room field* by the following equation (see figure 6-top):

$$RF_i(x, y, t) = 1 + k_r - k_r e^{-h_r((x-X_R^i)^2+(y-Y_R^i)^2)}$$

$$k_r, h_r > 0; \qquad 1 + k_r - k_r e^{-2h_r} \approx 1$$

The *crowd field* measures the amount of crowd in a room. It is evaluated by the infrastructure manipulating the *person presence* fields of the people in the building. The analytical description of this field is a bit more complicated than the previous ones, because it tries to abstract an entity that is strictly coupled with the discrete nature of the space considered by means of a continuous field.

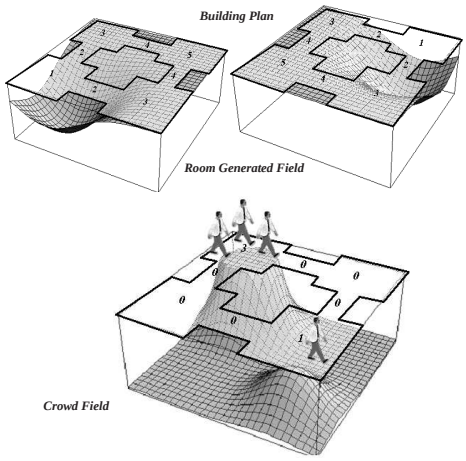


**Fig. 6.** (top) Room Fields and (bottom) Crowd Field in a building.

The crowd field, in fact, should in principle have a constant value in the area of a room and drop to zero (or to another value) outside that area. However, in order to avoid discontinuities, the crowd field, generated by the room $i$ at the coordinates $(X_R^i, Y_R^i)$, can be modeled by means of the following function (see figure 6-bottom).

$$CWF_i(x, y, t) = k_c^i e^{-h_c((x-X_R^i)^2+(y-Y_R^i)^2)}$$

Where $h_c$ is chosen so as the magnitude of the field outside the room is almost 0, while $k_c^i > 0$ is a coefficient that represents how much crowded the room is. It is important to understand, that because of its role, $k_c^i$ cannot be a fixed coefficient, but it must vary over time to reflect the different crowd conditions, so in general $k_c^i = k_c^i(t)$. $k_c^i(t)$ is simply given by the number of people present in room $i$ at time $t$ (whose coordinates are within the room's perimeter) normalized to the dimension of the room. It is clear that the $k_c^i(t)$ defined in this way is an effective measure of how much a room is crowded. Finally the global *crowd field* will be defined by means of the sum of the rooms' *crowd fields*:

$$CWF(x, y, t) = \sum_{i \in rooms} CWF_i(x, y, t)$$

Let's start considering the load balancing coordination protocol, whose aim is to help a user to avoid crowds or lines while roaming through the building. Basically, each user specifies his/her path through the building by selecting the rooms he/she wants to visit. This can be accomplished by letting the users' agents evaluate their *coordination field* as a minimum combination (fields are combined by taking in each point the minimum one) of the *room fields (RF)* to which their users are interested.

$$CF(x,y,t) = min(RF_i(x,y,t) : i = 1,2,...,n)$$

The *coordination field* produced models a surface having minimum points in correspondence of the rooms the user wants to visit. The user follows greedily the decrease of the *coordination field* and thus visiting the associated rooms. In order not to get trapped in a minimum, when the user completed the visit of a room, the corresponding field is removed from the combination and so it does not represent a minimum anymore. To take into consideration the load balancing service, the *crowd field (CWF)* has to be taken into account and for this purpose the combination computed by the agent can become:

$$CF(x,y,t) = min(RF_i(x,y,t) : i = 1,2,...,n) + \mu CWF(x,y,t)$$

The term $\mu CWF(x,y,t)$ with $\mu \geq 0$ is a field that has its maximum points where the crowd is particularly intense, as depicted in figure 6-bottom. This term manages the load balancing policy: in fact when this term is added to the minimum combination it changes the steepness of the *coordination field* in the crowded zones. In particular a crowded zone tends to be a peak in the *coordination field* and thus it tends to repulse the income of other agents. It is easy in fact to understand that the agent will follow the "greed path" - the one indicated by $min(RF_i(x,y,t) : i = 1,2,...,n)$- only if $CF$ decreases towards the same direction indicated by $min(RF_i(x,y,t) : i = 1,2,...,n)$. Indicating this direction with $v$, this condition can be restated as:

$$\nabla(min(RF_i(x,y,t) : i = 1,2,...,n) + \mu CWF)(v) \leq 0 \Rightarrow$$

$$\mu \leq -\frac{\nabla min(RF_i(x,y,t) : i = 1,2,...,n)(v)}{\nabla CWF(v)}$$

Observe that $\nabla min(RF_i(x,y,t) : i = 1,2,...,n)(v) \leq 0$ and thus the condition $\mu \geq 0$ can be satisfied. For this reason $\mu$ can be regarded as a term specifying the relevance of the *crowd field*. If $\mu$ is too high the agent will suggest the user to follow alternative (possibly longer) uncrowned paths towards the destination whenever the "greed" one will be a bit crowded. If $\mu$ is too low the agent will accept always to remain in the "greed" path disregarding the traffic conditions. The result of the numerical integration of the differential equations is depicted in figure 7(a). Here an agent is willing to proceed from $(0,0)$ to $(10,0)$. If it does not consider crowd (figure 7(a)-left) it follows a straight line form source to destination (eventually queuing in the middle). Otherwise, it is able to avoid the crowded area in the middle, by walking around it (figure 7(a)-right). Figure 7(b) shows the result of the motion coordination in two different simulated environments. On the left the crowd is not considered in the agents'*coordination field* and thus crowds and queues arise. On the right the crowd term is considered and thus crowds are avoided.

### 4.4 Room Field and Crowd Field: Meetings

Let's turn the attention to a "meeting" protocol whose aim is to help a group of users to dynamically find and move towards the most suitable room for a meeting.

The definition of this coordination protocol can rely on the previously introduced field by changing the *coordination field* perceived by agents. Also, this coordination protocol can be built upon the previous coordination protocol so as to retain the load-balanced movements also when agents are moving to meet each other. Several different protocols can be though related to how a group of agents should meet:

1. The group of users wants to meet in a particular room $x$. This is the simplest case and each of the user has to compute the following *coordination field*.

$$CF(x, y, t) = RF_x(x, y, t) + \mu CWF(x, y, t)$$

   In this way every agent is directed to the minimum of that leads to the meeting room. The crowd field term enforces the load balancing policy also in this case.
2. The group of users wants to meet in the room where person $x$ is located. This is very simple as well: each of the user has to compute the following *coordination field*

$$CF(x, y, t) = PRES_x(x, y, t) + \mu CWF(x, y, t)$$

   Where $PRES_x$ is the field generated by person $x$. In this way every agent is directed to the minimum of $PRES_x$ that leads to the meeting room (where person $x$ is located). It is interesting to notice that this approach works even if person $x$ moves after the meeting has been scheduled. The meeting will be automatically rescheduled in the new minimum of $PRES_x$.
3. The group of users wants to meet in the room that is between them (their "barycenter"). To this purpose each user i can compose its *coordination field* by combining the fields of all the other users:

$$CF(x, y, t) = \sum_{i \neq x} PRES_x(x, y, t) + \mu CWF(x, y, t)$$

   In this way all the users "fall" towards each other, and they meet in the room that is in the middle. It is interesting to notice, that this "middle room" is evaluated dynamically and the process takes into consideration the crowd. So if a room is overcrowded it will not be chosen, even if it is the room, which is physically in the middle of the users. For the same reason if some users encounter a crowd in their path to the meeting room, the meeting room is automatically changed to one closer to these unlucky users. The strength of this approach is that it is fully integrated with the field concept and that the meeting room is chosen to encounter the difficulties found by the user in real time.

By consider the third of the above three possibilities, the dynamical system description for this problem is straightforward and the integration of the differential equations is depicted in figure 8(a), showing how agents effectively converge to each other in the barycenter in an open space. Specifically, it depicts the x-y plane where agents live, with the trajectories of the agents of the system (i.e. the solutions of $(x_i(t), y_i(t))$ evaluated for a certain time interval). Figure 8(b) shows the various stages in the meeting process in two different simulated environments.

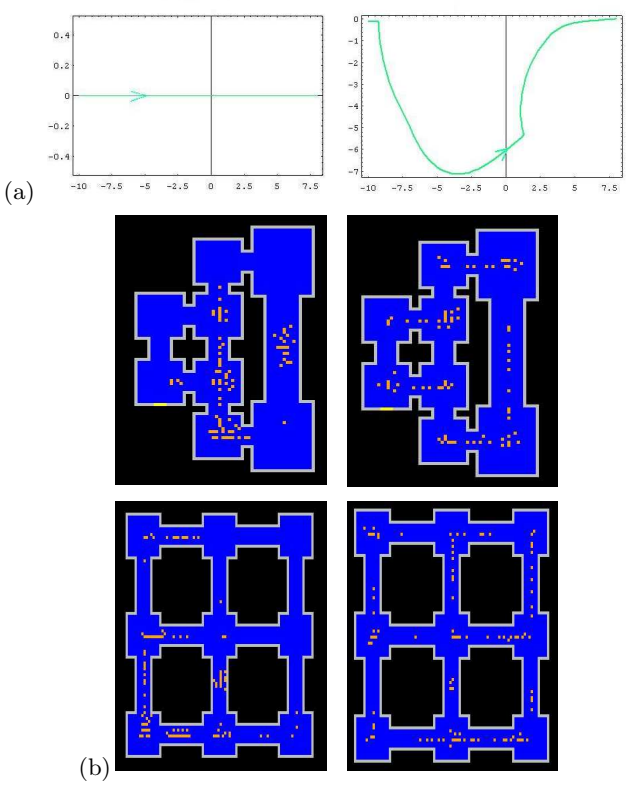


**Fig. 7.** (a) the trajectory followed by an agent when no crowd is encountered (left). Following downhill the *coordination field* the agent avoids a crowd between itself and its target (right). (b) Load balancing in two different simulated museums. When agents do not exploit Co-Fields, crowded zones appear (left). These crowded zones are mostly avoided when agents follow the proper *coordination field* (right).

## 5 Programming Motion Coordination Protocols

Turning the discussion more concrete: what software architecture is required to support field-based protocols? How can be implemented? How can be programmed?

The following of this section will focus on field based protocols implemented through distributed data structures actually spread in the environment.

### 5.1 Architecture and Implementation of a Field-based System

The middleware considered in this section, to support field-based coordination is Tuples On The Air (TOTA) [19]. TOTA proposes relying on distributed tuples to represent fields. Tuples are not associated to a specific node (or to a specific data space). Instead, tuples are injected in the network and can autonomously propagate and diffuse across the network accordingly to a specified pattern. Thus, TOTA tuples form spatially distributed data structures suitable to represent fields. To support this idea, TOTA is composed by a peer-to-peer network of possibly mobile nodes, each running a local version of the TOTA middleware. Upon the distributed space identified by the dynamic network of TOTA nodes, each component is capable of locally storing tuples and letting them diffuse through the network. Tuples are injected in the system from a particular node, and spread hop-by-hop accordingly to their propagation rule. In fact, a TOTA tuple is defined in terms of a "content" and a "propagation rule".

*T=(C,P)*

The content $C$ is an ordered set of typed fields representing the information carried on by the tuple. The propagation rule $P$ determines how the tuple should be distributed and propagated across the network. This includes determining the "scope" of the tuple (i.e. the distance at which such tuple should be propagated and possibly the spatial direction of propagation) and how such propagation can be affected by the presence or the absence of other tuples in the system. In addition, the propagation rule can determine how the tuple's content should change while it is propagated.

The spatial structures created by tuples propagation must be maintained coherent despite network dynamism. To this end, the TOTA middleware supports tuples propagation actively and adaptively: by constantly monitoring the network local topology and the income of new tuples, the middleware automatically re-propagates tuples as soon as appropriate conditions occur. For instance, when new nodes get in touch with a network, TOTA automatically checks the propagation rules of the already stored tuples and eventually propagates the tuples to the new nodes. Similarly, when the topology changes due to nodes' movements, the distributed tuples automatically change to reflect the new topology (see figure 9).

From the architecture point of view, each TOTA middleware is constituted by three main parts (see figure 10): *(i)* the TOTA API, is the main interface to access the middleware. It provides functionalities to let the application to inject new tuples in the system (*inject* method), to read tuples both from the local tuple

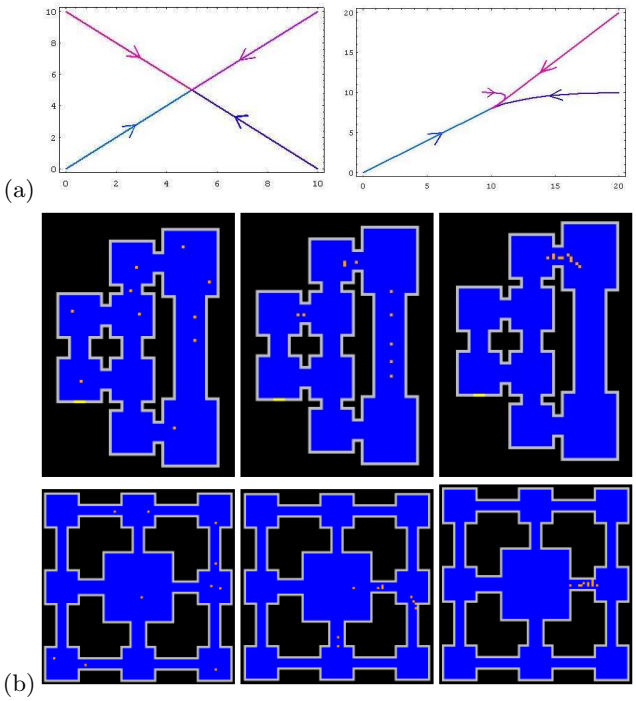


**Fig. 8.** (a) Meeting: Solutions of the system's differential equations, for different initial conditions. (b) from left to right different stages in the meeting process, showing that agents converge towards each other.

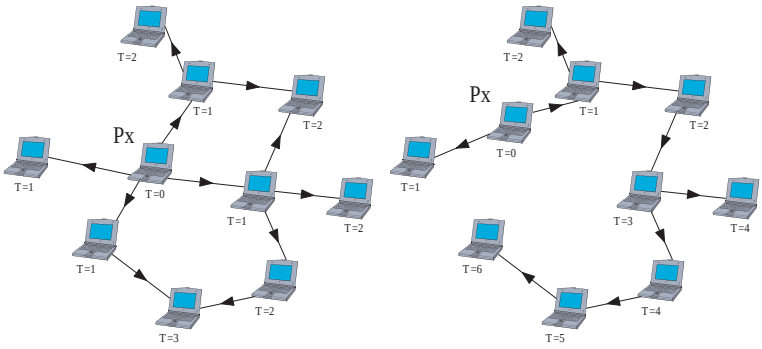


**Fig. 9.** (left) Px propagates a tuple that increases its value by one at every hop. (right) when the tuple source Px moves, all tuples are updated to take into account the new topology

space and from the node's one-hop neighborhood, either via pattern matching or via key-access to a tuple's unique id (*read, readOneHop, keyrd, keyrdOneHop* methods), to delete tuples from the local middleware (*delete* method), or to place and remove subscriptions in the event interface (*subscribe, unsubscribe* methods). Moreover, two methods allow tuples to be actually stored in he local tuple space or to be propagated to neighboring nodes (*store, move* methods) *(ii)* The EVENT INTERFACE is the component in charge of asynchronously notifying the application about subscribed events. Basically upon a matching event, the middleware invokes on the subscribed components a *react* method to handle the event. *(iii)* The TOTA ENGINE is the core of TOTA: it is in charge of maintaining the TOTA network by storing the references to neighboring nodes and to manage tuples' propagation by executing their propagation methods and by sending and receiving tuples. Within this component is located the tuple space in which to store TOTA tuples.

From an implementation point of view, a first prototype of TOTA running on Laptops and on Compaq IPAQs equipped with 802.11b, Familiar LINUX and J2ME-CDC (Personal Profile) has been developed. IPAQs connect locally in the MANET mode (i.e. without requiring access points) creating the skeleton of the TOTA network. Tuples are being propagated through multicast sockets to all the nodes in the one-hop neighborhood. The use of multicast sockets has been chosen to improve the communication speed by avoiding 802.11b unicast handshake. By considering the way in which tuples are propagated, TOTA is very well suited for this kind of broadcast communication. This is a very important feature, because it could allow implementing TOTA also on really simple devices (e.g. micro sensors) that cannot be provided with sophisticate (unicast enabled) communication mechanisms. Other than this communication mechanism, at the core of the TOTA middleware there is a simple event-based engine, that monitors network reconfigurations and the income of new tuples and react either by triggering propagation of already stored tuples or by generating an event directed to the event interface.

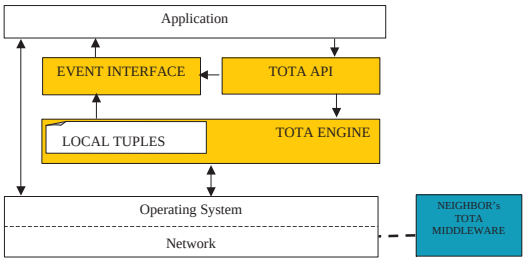


**Fig. 10.** TOTA middleware architecture

### 5.2 Specifying Tuples and Agents

Relying on an object oriented methodology, TOTA tuples are actually objects: the object state models the tuple content, while the tuple's propagation has been encoded by means of a specific *propagate* method.

When a tuple is injected in the network, it receives a reference to the local instance of the TOTA middleware, then its code is actually executed (the middleware invokes the tuple's *propagate* method). If during the tuple's execution, the middleware *move* method is called, the tuple is actually sent to all the one-hop neighbors, where it will be executed recursively. During migration, the object state (i.e. tuple content) is properly serialized to be preserved and rebuilt upon the arrival in the new host.

Following this schema, It is possible to define an abstract class TotaTuple, that provides a general framework for tuples programming. The code of this class is in the following:

```
abstract class TotaTuple {
 protected TotaInterface tota;
 /* the state is the tuple content */
 /* this method inits the tuple, by giving a reference
 to the current TOTA middleware */
 public void init(TotaInterface tota) {
  this.tota = tota;
 }
 /* this method codes the tuple actual actions */
 public abstract void propagate();
 /* this method enables the tuple to react
 to happening events see later in the article */
 public void react(String reaction, String event)
{}}
```

It is worth noting that in TOTA, a tuple does not own a thread, but it is actually executed by the middleware that runs the tuple's *init* and *propagate* methods. Tuples, however, must remain active even after the middleware has run their code. This is fundamental because their self-maintenance algorithm, for example, must be executed whenever the right condition appears (e.g. when a new node connects to the network, the tuples must propagate to this newly arrived node). To this end, tuples can place subscriptions as provided by the standard TOTA API. These subscriptions let the tuples remain "alive", being able to execute upon triggering conditions.

A programmer can obtain new tuples by subclassing the *TotaTuple* class. Moreover, to facilitate the creation of tuples, a tuples' class hierarch from which the programmer can inherit has been developed. The class hierarchy enable a programmer to create custom tuples without worrying about most of all the intricacies of dealing with tuple propagation and maintenance (a piece of the hierarchy is depicted in figure 11). More detailed information about the complete class hierarchy can be found in [20].

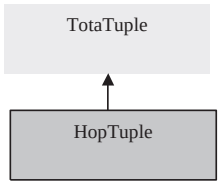


**Fig. 11.** Tuples' class hierarchy.

The class *HopTuple* inherits from *TotaTuple*. This class is a template to create self-maintained distributed data structures over the network. Specifically, it implements the superclass method *propagate*, as shown in the following:

```
public final void propagate() {
 if(decideEnter()) {
  boolean prop = decidePropagate();
  changeTupleContent();
  this.makeSubscriptions();
  tota.store(this);
  if(prop)
   tota.move(this);
}}
```

In particular, this propagation method is realized by executing a sequence of other specific methods: *decideEnter*, *decidePropagate*, *changeTupleContent* and *makeSubscriptions* so as to realize a breadth first, expanding ring propagation. The result is simply a tuple that floods the network without changing its content:

- When a tuple arrives in a node (either because it has been injected or it has been sent from a neighbor node) the middleware executes the *decideEnter* method that returns true if the tuple can enter the middleware and actually execute there, false otherwise. The standard implementation returns true if the middleware does not already contain that tuple.
- If the tuple is allowed to enter the method *decidePropagate* is run. It returns true if the tuple has to be further propagated, false otherwise. The standard implementation of this method returns always true, realizing a tuple's that floods the network being recursively propagated to all the peers.
- The method *changeTupleContent* changes the content of the tuple. The standard implementation of this method does not change the tuple content.
- The method *makeSubscriptions* allows the tuple to place subscriptions in the TOTA middleware. As stated before, in this way the tuple can react to events even when they happen after the tuple completes its execution. The standard implementation does not subscribe to anything.
- After that, the tuple is inserted in the TOTA tuple space by executing *tota.store(this)*.

– Then, if the *decidePropagate* method returned true, the tuple is propagated to all the neighbors via the command *tota.move(this)*. The tuple will eventually reach neighboring nodes, where it will be executed again. It is worth noting that the tuple will arrive in the neighboring nodes with the content changed by the last run of the *changeTupleContent* method.

Programming a TOTA tuple to create a distributed data structure basically reduces at inheriting from the above class and overloading the four above methods to customize the tuple behavior. Here in the following, two examples (*Gradient* and *Flock* tuples) are presented to clarify the TOTA framework.

A *Gradient* tuple creates a tuple that floods the network in a breadth-first way and have an integer hop-counter that is incremented by one at every hop. *Flock* tuple are slightly more complicated in that they overload the *changeTupleContent* method to create the flocking shape. The actual code of these tuples is in the following:

```
public class Gradient extends HopTuple {
 public String name;
 public int val = 0;

 protected void changeTupleContent() {
 /* The correct hop value is maintained
 in the superclass HopBasedTuple */
 super.changeTupleContent();
 val = hop;
}}


public class FlockTuple extends HopTuple {
 public String peerName;
 public int val = 2;
 /* always propagate */
 public HopBasedTuple changeTupleContent() {
 super.changeTupleContent();
 /* The correct hop value is maintained in
 the superclass HopBasedTuple */
 if(hop <= 2) val--;
  else if(hop >2) val++;
 return this;
}}
```

It is rather easy now to program the agents required in the case study. Let's focus the attention on the meeting coordination protocol described in 4.4, in the case in which agents are going to meet in their barycenter room. With this regard, the algorithm followed by meeting agents is very simple: agents have to determine the furthest agent, and then move by following downhill that

agent's presence field (implemented by means of a *Gradient* tuple). In this way agents will meet in their barycenter. More in detail, at start-up each agent will propagate its presence field, by injecting the associated tuple. Then it will read its local tuple space to determine the farthest agent (the one whose presence field is bigger). Then it will inspect its one-hop neighborhood to find the local shape of the presence fields of the selected agent. Finally, it will move by following the tuple's gradient downhill. More information on this and on the problems that arise can be found in [20]. The actual code is in the following:

```
public class MeetingAgent extends Thread implements AgentInterface
{
 private TotaMiddleware tota;

 public void run() {
  /* inject the meeting tuple to participate the meeting */
  Gradient mt = new Gradient ();
  mt.setContent(peer.toString());
  tota.inject(mt);

  while(true) {
   /* read other agents' meeting tuples */
   Gradient presence = new Gradient();
   Vector v = tota.read(presence);
   /* evaluate the gradients and select the peer to
   which the gradient goes downhill */
   Gradient farther = getMaximum(v); Vector w = tota.
   KeyrdOneHop(farther);
   GenPoint destination = getDestination(w);
   /* move downhill following the meeting tuple */
   peer.move(destination);
}}}
```

## 6 Open Issues and Research Directions

This chapter presented the field-based approach to coordinate the movements of a large number of autonomous agents in a wireless and mobile computing scenario. At the moment the main weakness of this approach is the lack of a general methodology to help identifying, given a specific motion pattern to be enforced, which fields have to be defined, how they should be propagated, and how they should be combined in a *coordination field*. Nevertheless, the immediate applicability of field-based protocols is guaranteed by the possibility of getting inspiration from (and of reverse engineering) a wide variety of motion patterns present in nature. Phenomena such as diffusion, birds' flocking, ants' foraging, bee dances [4], to mention a few examples, can all be easily modeled with fields (i.e., in terms of agents climbing/descending a *coordination field* obtained as the composition of some computational fields), and all have practical

application in mobile computing scenarios. However, it is likely that a complete general methodology could be found in the future and it would allow to develop sound engineering procedures to develop field-based protocols. In pursuing this long-term goal, deployment of applications will definitely help identifying current shortcomings and directions of improvement. With this regard, particularly significant results can possibly be obtained by understanding how the field approach could encode coordination tasks that don't deal with physical movements, but with other kind of actions. In these cases, a *coordination field* can be thought as spread in an abstract coordination space to encode any kind of coordination actions. Agents would be driven by the fields, not by moving form one place to another, but by making other kind of activities. In this direction, in the next future it will be interesting to apply the field-based model, in the development of new applications for sensor networks with a particular interest in those algorithms exploiting ideas taken from manifold geometry [31].

## References


1. H. Abelson, et al., "Amorphous Computing", Communications of the ACM, 43(5), May 2000.
2. S. Bandini, S. Manzoni, C. Simone, "Heterogeneous Agents Situated in Heterogeneous Spaces", 3rd International Symposium From Agent Theories to Agent Implementations, Vienna (A), April 2002.
3. F. Bellifemine, A. Poggi, G. Rimassa, "JADE - A FIPA2000 Compliant Agent Development Environment", 5th International Conference on Autonomous Agents (Agents 2001), Montreal (CA), May 2001.
4. E. Bonabeau, M. Dorigo, G. Theraulaz, "Swarm Intelligence", Oxford University Press, 1999.
5. W. Butera, "Programming a Paintable Computer", PhD Thesis, MIT Media Lab, Feb. 2002.
6. G. Cabri, L. Leonardi, F. Zambonelli, "Engineering Mobile Agent Applications via Context-dependent Coordination", IEEE Transaction on Software Engineering, 28(11):1040-1056, Nov. 2002.
7. A. Carzaniga, D. Rosenblum, A. Wolf, "Design and Evaluation of a Wide-Area Event Notification Service", ACM Transactions on Computer Systems, 19(3), pp. 332-383, aug 2001.
8. G. Cugola, A. Fuggetta, E. De Nitto, "The JEDI Event-based Infrastructure and its Application to the Development of the OPSS WFMS", IEEE Transactions on Software Engineering, 27(9): 827-850, Sept. 2001.
9. D. Estrin, D. Culler, K. Pister, G. Sukjatme, "Connecting the Physical World with Pervasive Networks", IEEE Pervasive Computing, 1(1):59-69, Jan. 2002.
10. E. Freeman, S. Hupfer, K. Arnold, "JavaSpaces Principles, Patterns, and Practice", Addison-Wesley, 1999.
11. J. Hightower, G. Borriello, "Location Systems for Ubiquitous Computing", IEEE Computer, 34(8): 57-66, Aug. 2001.
12. A. Howard, M. Mataric, G. Sukhatme, "An Incremental Self-Deployment Algorithm for Mobile Sensor Networks", Autonomous Robots,special issue on Intelligent Embedded Systems, G. Sukhatme, ed., 2002, 113-126.
13. Jini, http://www.jini.org, Sun Microsystems.

14. B. Johanson, A. Fox, "The Event Heap: A Coordination Infrastructure for Interactive Workspaces", 4th IEEE Workshop on Mobile Computer Systems and Applications (WMCSA-2002), Callicoon, New York, USA, June, 2002.
15. S. Johansson, A. Saffiotti, "Using the Electric Field Approach in the RoboCup Domain", Proceedings of the Int. RoboCup Symposium. Seattle, WA, 2001.
16. O. Khatib, "Real-time Obstacle Avoidance for Manipulators and Mobile Robots", The International Journal of Robotics Research, 5(1):90-98, 1986.
17. J. Kephart, D. M. Chess, "The Vision of Autonomic Computing", IEEE Computer, 36(1):41-50, Jan. 2003.
18. M. Mamei, F. Zambonelli, L. Leonardi, "Co-Fields: A Physically Inspired Approach to Distributed Motion Coordination". IEEE Pervasive Computing, 3(2):52-61, 2004.
19. M. Mamei, F. Zambonelli, L. Leonardi, "Tuples on the Air: a Middleware for Context-aware Computing in Dynamic Networks", Proceedings of the Workshops at the 23rd International Conference on Distributed Computing Systems, IEEE CS Press, Providence (RI), pp. 342-347, May 2003.
20. M. Mamei, F. Zambonelli, "Self-Maintained Distributed Data Structure Over Mobile Ad-Hoc Network", Technical Report No. DISMI-2003-23, University of Modena and Reggio Emilia, August 2003.
21. C. Mascolo, L. Capra, W. Emmerich, "An XML based Middleware for Peer-to-Peer Computing", In Proc. of the IEEE International Conference of Peer-to-Peer Computing (P2P2001), Linkoping, Sweden, Aug. 2001.
22. Mathematica, http://www.wolfram.com.
23. R. Nagpal, A. Kondacs, C. Chang, "Programming Methodology for Biologically-Inspired Self-Assembling Systems", in the AAAI Spring Symposium on Computational Synthesis: From Basic Building Blocks to High Level Functionality, March 2003
24. H. V. Parunak, S. Brueckner, J Sauter, "ERIM's Approach to Fine-Grained Agents", NASA Workshop on Radical Agent Concepts, Greenbelt, MD, USA, Jan. 2002.
25. G. P. Picco, A. L. Murphy, G. C. Roman, "LIME: a Middleware for Logical and Physical Mobility", 21st International Conference on Distributed Computing Systems, IEEE CS Press, pp. 524-536, July 2001.
26. Poor, "Embedded Networks: Pervasive, Low-Power, Wireless Connectivity", PhD Thesis, MIT, 2001.
27. D. Servat, A. Drogoul, "Combining amorphous computing and reactive agent-based systems: a paradigm for pervasive intelligence?", AAMAS, Bologna (I), July, 2002.
28. W. Shen, B. Salemi, P. Will, "Hormone-Inspired Adaptive Communication and Distributed Control for CONRO Self-Reconfigurable Robots", IEEE Transactions on Robotics and Automation 18(5):1-12, Oct. 2002.
29. The Sims, http://thesims.ea.com.
30. The Swarm Simulation Toolkit, http://www.swarm.org.
31. F. Zambonelli, M. Mamei, "The Cloak of Invisibility: Challenges and Applications", IEEE Pervasive Computing, 1(4):62-70, Oct.-Dec. 2002.
32. F. Zambonelli, V. Parunak, "From Design to Intentions: Sign of a Revolution", International Conference on Autonomous Agents and Multi-agent Systems, Bologna (I), July 2002.